Introduction to Robot Operating System (ROS) using C++

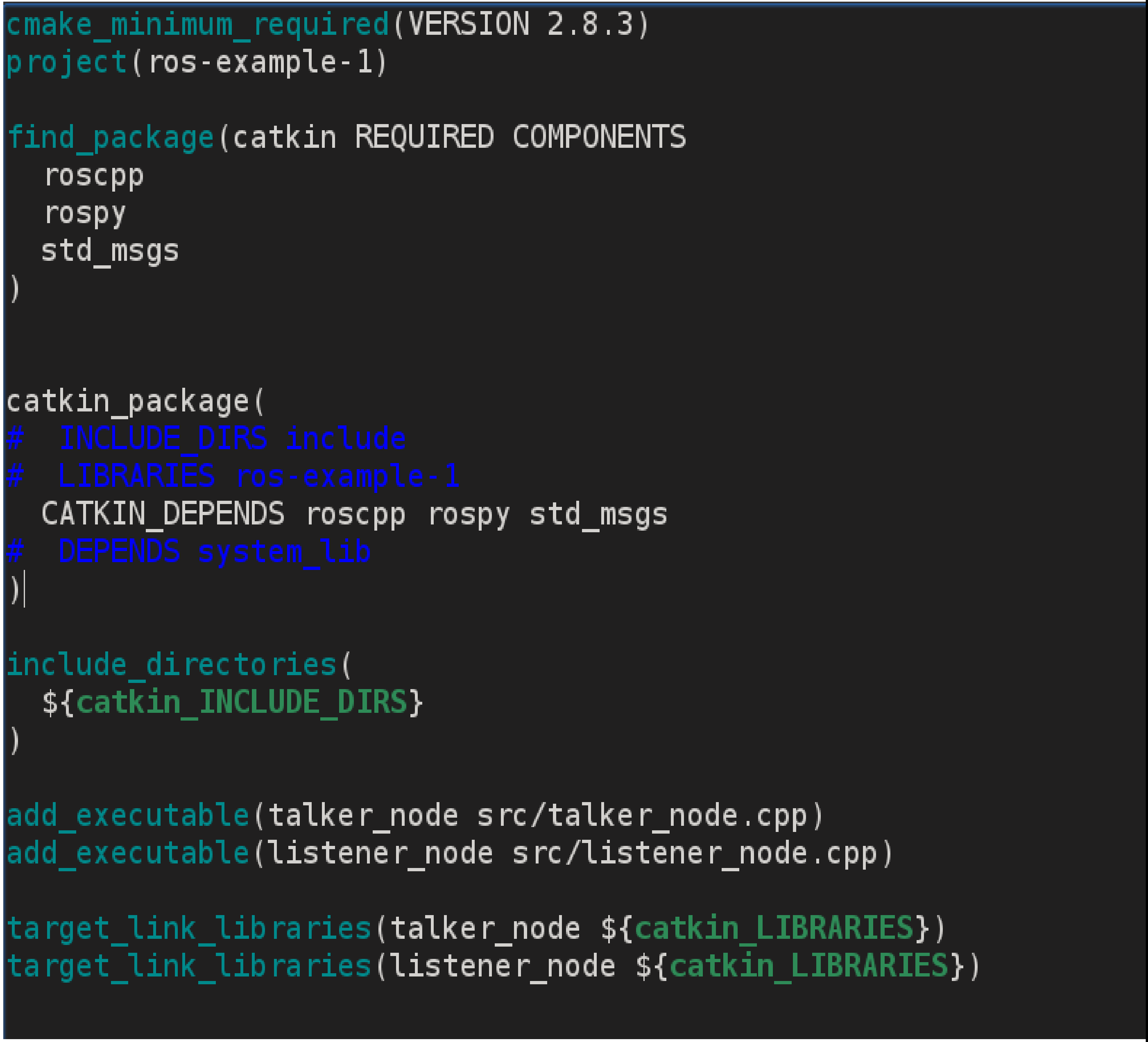
Mahmoud Abdul Galil

Tutorial-2

These slides are based on the online ROS Wiki documentation

ros-example-1 Cont'd: CMakeLists.txt

- We wrote our desired codes in src folder
- Now is time to inform the build system to include these codes while building our workspace.
- This is done by modifying the CMakeLists.txt file.
- Excluding documentation comments, the new CmakeLists.txt will look like the picture on the right
- Notice that we added two build targets as executables using **add_executable( )**, and linked catkin_LIBRARIES to then using **target_link_libraries( )**.



```
cmake_minimum_required(VERSION 2.8.3)
project(ros-example-1)

find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
)


catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES ros-example-1
  CATKIN_DEPENDS roscpp rospy std_msgs
#  DEPENDS system_lib
)

include_directories(
  ${catkin_INCLUDE_DIRS}
)

add_executable(talker_node src/talker_node.cpp)
add_executable(listener_node src/listener_node.cpp)

target_link_libraries(talker_node ${catkin_LIBRARIES})
target_link_libraries(listener_node ${catkin_LIBRARIES})
```

Outline

- Overview of ROS
- ROS file system level
- ROS communication graph level
- ROS build-tool catkin
- ROS command-line tools
- ROS examples
- Publisher ↔ Subscriber example
- Server ↔ Client example
- Custom message example
- INTERMEDIATE: Publisher&Subscriber class example
- INTERMEDIATE: Action ↔ Client example

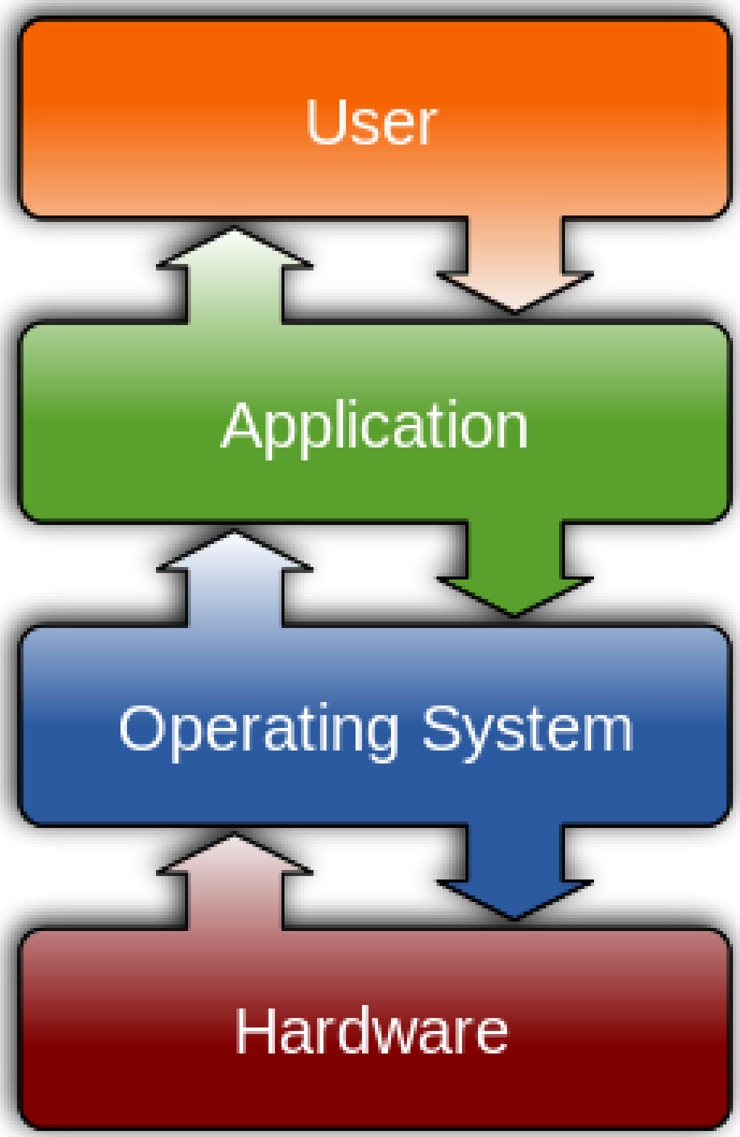
Overview of ROS

□ Concept of an Operating system:

□An operating system is a low-level software that manages computer hardware and software resources and provides common services to computer programs. All computer programs, except firmware, require the presence of an operating system[Wiki]. An OS is the only gate through which a computer application can interact with computer hardware.

□ ROS is a meta-operating system

□ROS is not an independent OS, it requires the presence of UNIX-like OS to work. However, it provides OS-like functions such as: inter-processes message passing, hardware abstraction, package management and other features.



User

Application

Operating System

Hardware

Overview of ROS cont'd

Real-time vs General-purpose operating systems

- RTOS differs from GPOS in the way the task scheduler works, in GPOS the scheduler manages resources to guarantee a certain goal that is often equal distribution of execution time. This leads to an unpredictable nature of the scheduler, thus a “non-deterministic” execution time. In an RTOS, the scheduler is designed to provide a predictable execution pattern, and thus a more-or-less a deterministic execution time

ROS ≠ RTOS

- ROS is not an RTOS, since it is relies on UNIX-like OSes, which are generally non-RTOS.

ROS + Orocos RTT = RTROS

- However, for hard-realtime requirements, integrations between ROS and third-party RTOS has been to provide real-time processing inside the ROS network, follow the link here if you want more details: http://www.orocos.org/rtt

Overview of ROS cont'd

- ROS is a middle-ware (glue-like code):

- In a complex system, such as a robot, many software applications/modules must co-exist and communicate efficiently in order for the system to function properly. A native GPOS such as windows or unix, generally, doesn't support an easy way to build such an ecosystem. Middle-wares exist to to facilitate such a task by providing API that allow to implement communications and input/output seamlessly among applications.

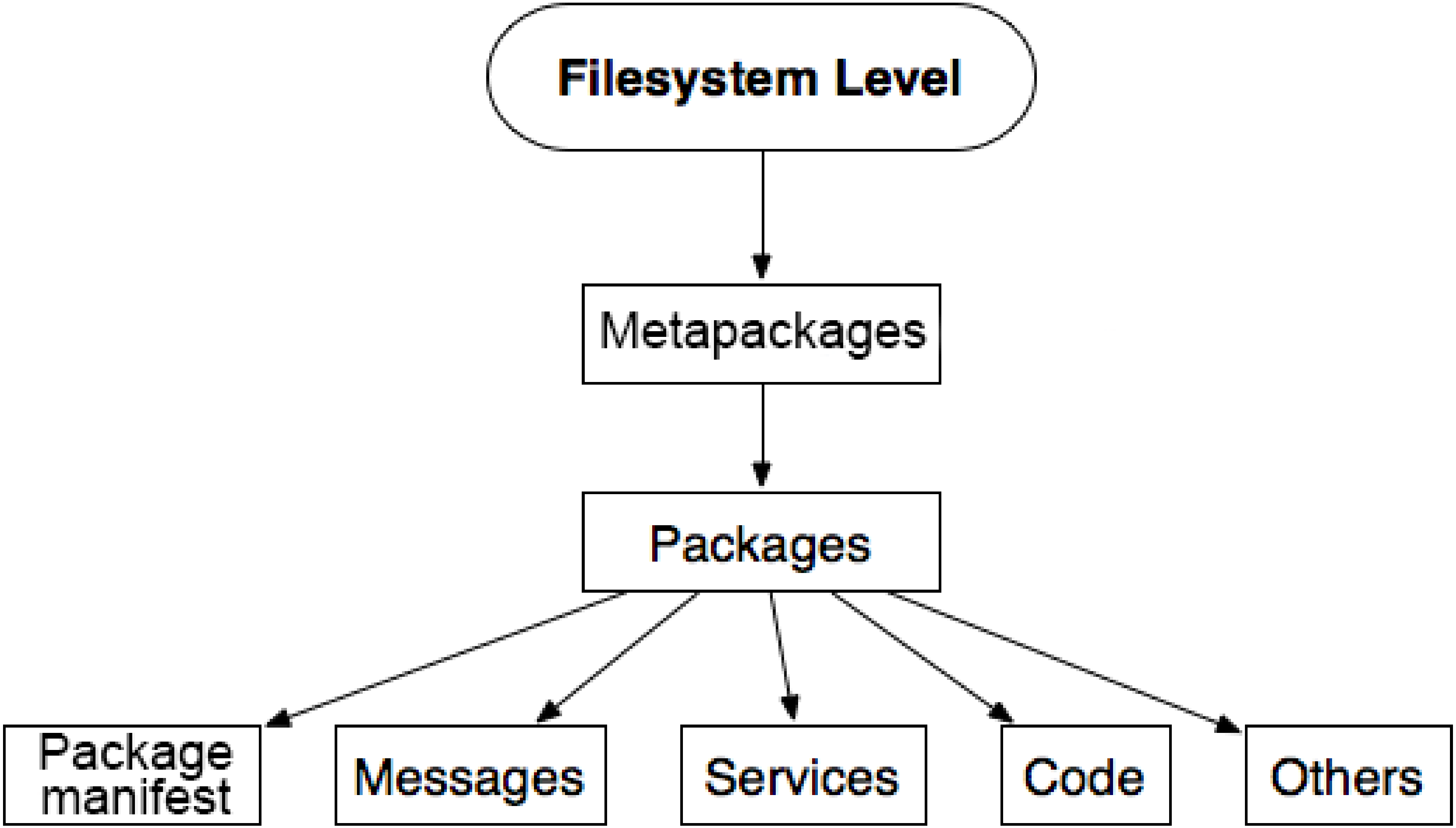
- ROS alternatives

- ROS is not the only middle-ware nor the oldest in robotics field. Many middle-wares such as **Orocos**, **Player/Stage**, **RT-middleware** are all alternatives to ROS. However, ROS true power lies in its conceptual design that promotes maximum code re-usability. In addition to flexible APIs that allow for seamless integration with other third-party libraries and applications, and most importantly, ROS adopts the open-source initiative, which may have been the main reason behind its sudden popularity nowadays.

- ROS community and ROS answers

- One of the strongest pros of ROS is the big and expanding online community. Virtually anyone can contribute to the ROS project and thousands of packages are available under opensource licenses for free on the internet. Once you have a functional ROS system on your machine, you can easily browse through the available softwares and pick whatever suits your application.

ROS Filesystem Level



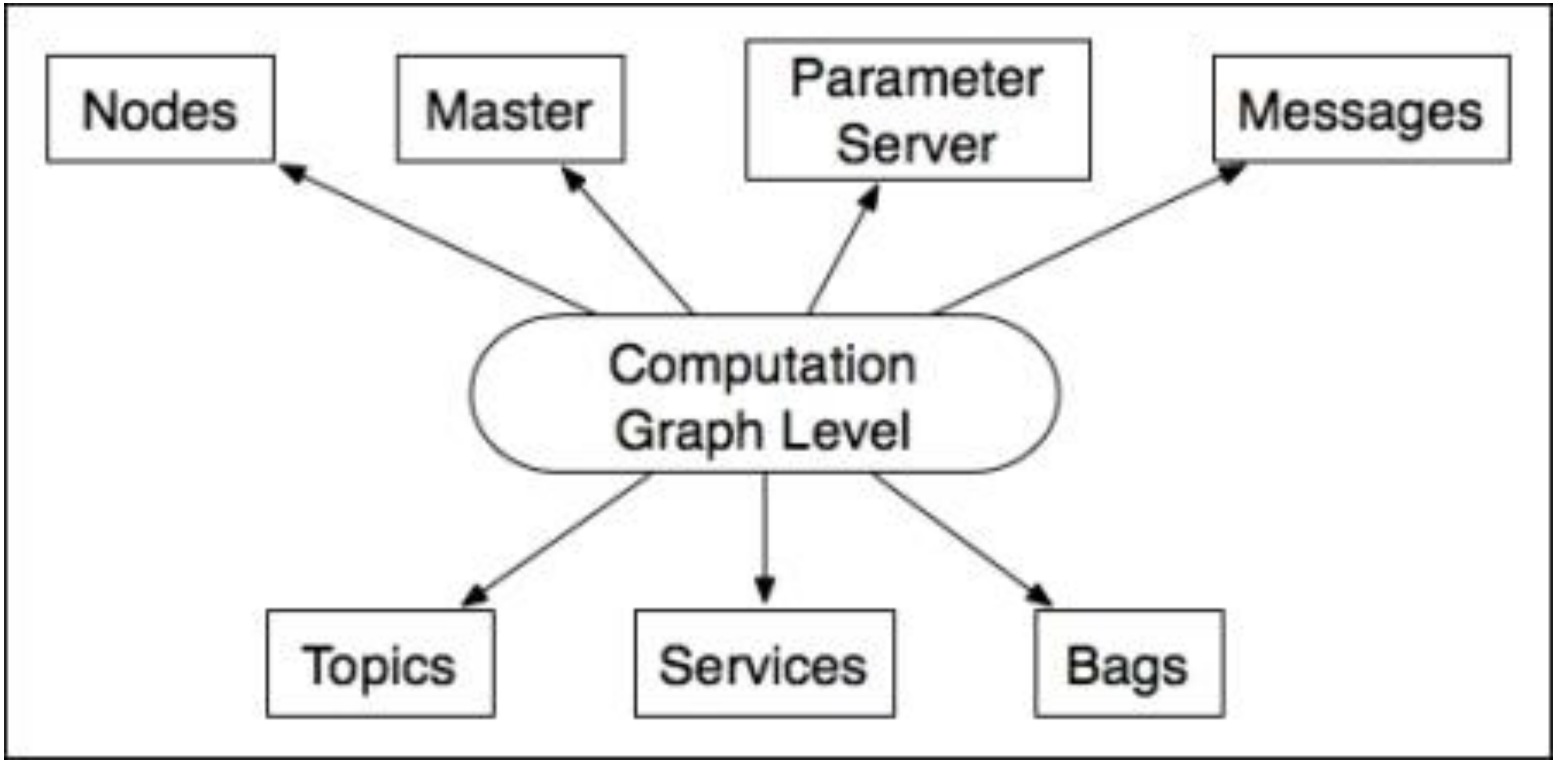
ROS Filesystem Level: Packages

- Packages are the basic building blocks of software in the ROS framework
- Packages can contain any-type of ROS-based software such as ROS-runtime processes (nodes), ROS-independent software or even third-party software
- Each package must contain a package.xml file, also called a manifest file, that describes the package and its build/runtime dependencies among other information
- Meta-packages are a bunch of related packages that do similar functions or serve the same target, grouped together
- It's only through packages that ROS-based software can be developed
- Check ROS-Wiki for more information: http://wiki.ros.org/Packages

ROS Client Libraries

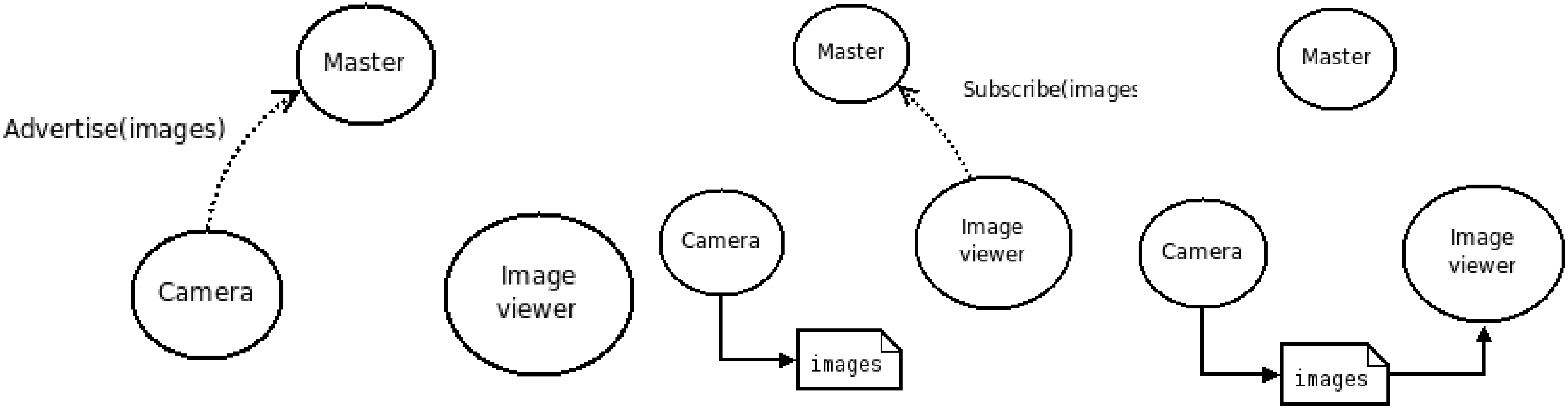
- ROS client libraries are the way to write ROS-enabled code
- Client libraries implement ROS concepts in APIs available for development
- Client libraries exist in many APIs, a few of them are: C++, Python, Lisp and Java
- We will be using roscpp client library to write ROS-enabled C++ code.

ROS Computation Graph Level



ROS Computation Graph Level: Master

- The ROSMASTER node is the main player in a ROS network.
- Its function is similar to that of a Domain Name Server (DNS)
- The role of the Master is to enable individual ROS nodes to locate one another.
- The ROSMASTER also hosts the Parameter Server
- More Info: http://wiki.ros.org/Master



ROS Computation Graph Level: Nodes

- Nodes are run-time processes in the ROS framework
- Nodes can be written in C++ or other client library provided APIs
- Instead of “monolithic” code approach, nodes provide a convenient way of distributing computations between several software modules, thus increasing fault tolerance and increasing debug-ability of code
- More Info: http://wiki.ros.org/Nodes

ROS Computation Graph Level: Messages

- A message is a simple data structure consisting of different data fields
- Nodes communicate with each other by publishing messages to topics.
- Message definition files (.msg) constitute a simple way of defining message data structures
- Message generation modules provided by client libraries are then used to generate code from .msg files
- More Info: http://wiki.ros.org/Messages

ROS Computation Graph Level: Topics

- Topics are named buses over which nodes exchange messages.
- Topics are asynchronous communication channels; the production of information is decoupled from its consumption.
- There can be multiple publishers and subscribers to a single topic
- More info: http://wiki.ros.org/Topics

A simplifying analogy for asynchronous communication in ROS (topic-based communication)

- Topic↔ Board
- Messages ↔ Notes
- Given the above abstractions, a publisher node can be abstracted as a person that hangs a note on the board, and a subscriber node can be abstracted as a person that is looking on the board observing any thing hanged on it.

ROS Computation Graph Level: Services

- Services are a strictly one-to-one communication scheme between nodes.
- Services are remote procedure calls
- A client node invokes a procedure call by sending a request to a remote server node
- The server node then replies with a response
- Server ↔ Client communication is a synchronous type of communication in ROS
- This means that a client node enters an idle state until the server node replies with the response
- If the server fails to respond within time, the communication fails

ROS Computation Graph Level: Actions

- Actions are similar to services in that they are another form of one-to-one synchronous communication channel in ROS
- Actions are different from services in that actions are interruptible, if the action server fails to respond before timeout, the client node preempts the action server, forcing it to respond with a predefined procedure
- Actions can be used to invoke procedure calls and monitor their progress.
- More Info: http://wiki.ros.org/actionlib

ROS Computation Graph Level: Bags

- Bags are an out-of-the-box support of ROS to provide record-and-play functionality for messages
- Bags can be used to record sensor readings and use play them back later on to simulate same situations without doing the hardware part all over
- More Info: http://wiki.ros.org/Bags

ROS Computation Graph Level: Parameter Server

□Parameter server is a part of the ROSMASTER

□It is a globally shared multi-variate dictionary, accessible to all nodes in a given ROS network through the network APIs.

□As it is not designed for high-performance, it is best used for static, non-binary data such as configuration parameters.

□More Info: http://wiki.ros.org/Parameter%20Server

ROS build-tool catkin

□What is a build-system?

□A build system is responsible for generating 'targets' from raw source code. These targets may be in the form of libraries, executable programs or anything that is not static code

□How to create a catkin workspace?

□A catkin workspace is the main directory in which ROS packages are created and built. Creating a catkin workspace is done by invoking the command catkin_init_workspace inside the src directory of the main workspace directory

□Creating and building ROS packages using catkin command-line tools

□A catkin package is created by invoking the command catkin_create_pkg inside the src directory

ROS build-tool catkin cont'd

- Some important catkin command line tools
  - catkin_init_workspace
    - $ mkdir catkin_ws & cd catkin_ws
    - $ mkdir src & cd src
    - $ catkin_init_workspace
  - catkin_create_pkg
    - $ catkin_create_pkg test_package roscpp std_msgs
  - catkin_make
    - $ catkin_make
    - Note: the last command must be invoked in the base directory of the catkin workspace

The mysterious “CMakeLists.txt” file

- Each catkin-enabled package must contain a CmakeLists.txt that instructs the compiler about the targets and build configuration
- The main tags in a CmakeLists.txt file for a catkin package are:
    - cmake_require_minimum
    - project
    - find_package
    - add_executable
    - add_library
    - target_link_libraries

ROS Command-line tools

- Rostopic
- Rosmsg
- Rosservice
- Roscd
- Rosrun
- Roslaunch
- Rospack
- Rosnode

- Roscore
- Rosparam
- Rosgraph
- Roswtf : (what the f…….fault)

ROS Examples: ros_example_1

- Publisher Node ==> talker_node.cpp
- Subscriber Node ==> listener_node.cpp
- CmakeLists.txt

ROS Examples: ros_example_2

- Service definition file (.srv)
- Server Node ==> server_node.cpp
- Client Node ==> client_node.cpp
- CmakeLists.txt

References

http://wiki.ros.org/ROS/Tutorials
https://github.com/qboticslabs/mastering_ros